Software Specifications

Interprocess Communications, Redstone dp3

Checkout and Launch Control System (CLCS)

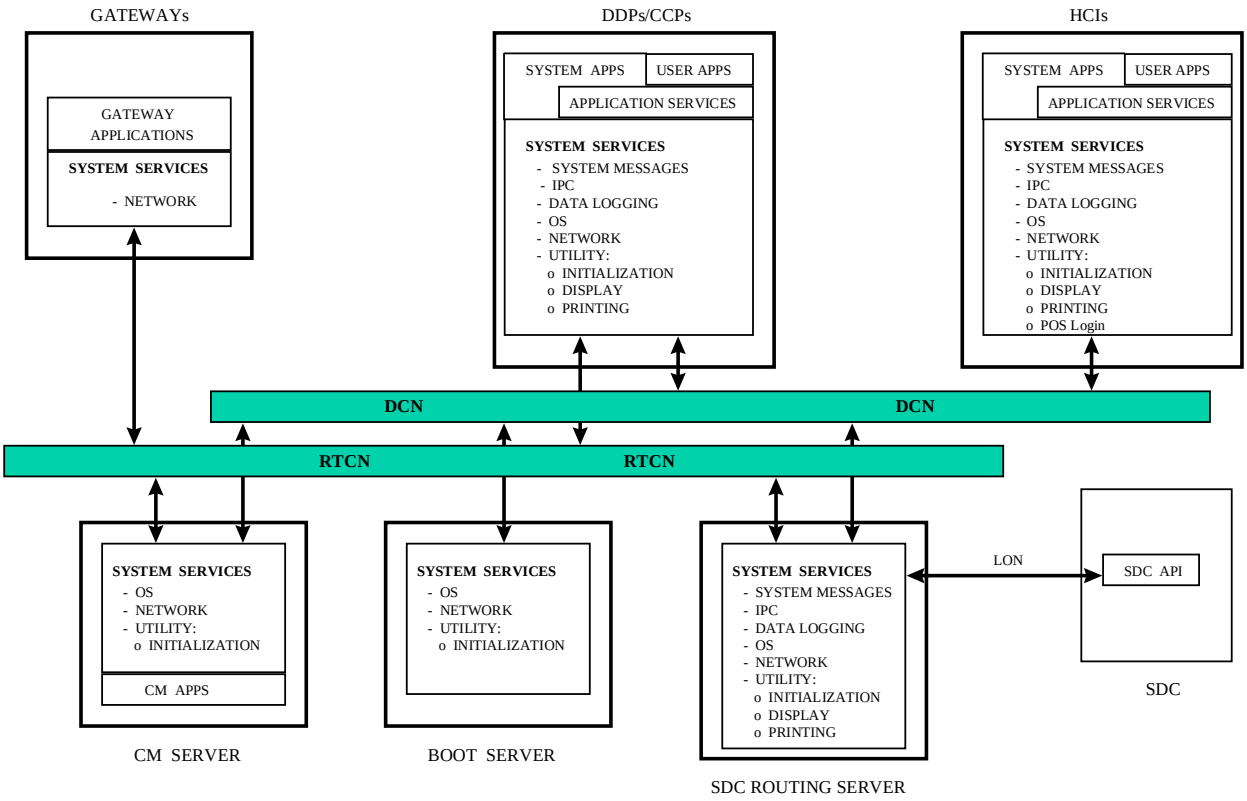
84K00510-070

1. IPC Services

1.1 IPC Services Introduction

1.1.1 IPC Services Overview

IPC Services (IPC) consists of a set of processes and API's that applications can utilize to receive both internode and intranode events (messages). IPC provides location transparency, which means it does not require the sender to know the platform residence of the message receiver. For Redstone, the IPC API's will be enhanced to generate the Packet Payload based on parameters sent by the application and conduct its distribution.



SYSTEM SERVICES OVERVIEW

1.1.2 IPC Services Operational Description

Applications register with IPC Services via an API call. After registration, applications can send and receive messages. The delivery message type determines intra-node or inter-node communications. Messages are routed, via API 's, to other applications.

The sending application can also specify whether it wants to block or not in waiting for an acknowledgement or negative acknowledgment from a send request. An application may specify the time in milliseconds to wait for the entire send operation to complete. Two types of delivery methods are supported:

- Intra-Node Communication

Intra-Node communication for message distribution between applications executing within the same platform. Note, if the record option is requested, the message is set out on the network to the SDC. Refer to Figure 1.

- Inter-Node Communication

  Inter-Node communication for message distribution between applications executing on different platforms. Refer to Figure 2.

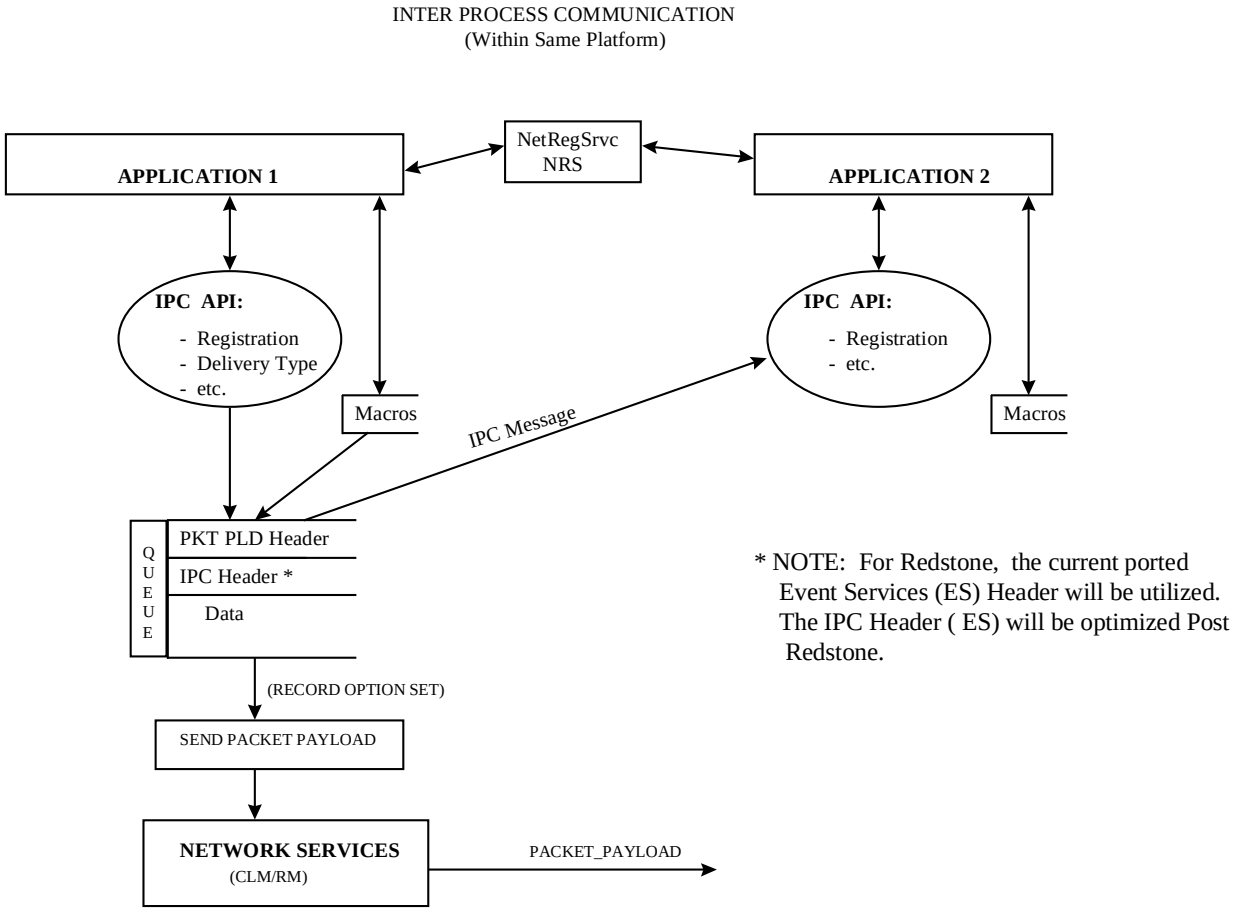


Figure 1. Intra-Node Communications

Application processes are able to receive messages via the programmatic interfaces. The receiving process can specify whether it wants to block or not for receipt of a message. If the receiving application chooses to block, it may specify the time in milliseconds to wait for an event to be received. Messages will be received in priority order at the priority specified by the sender

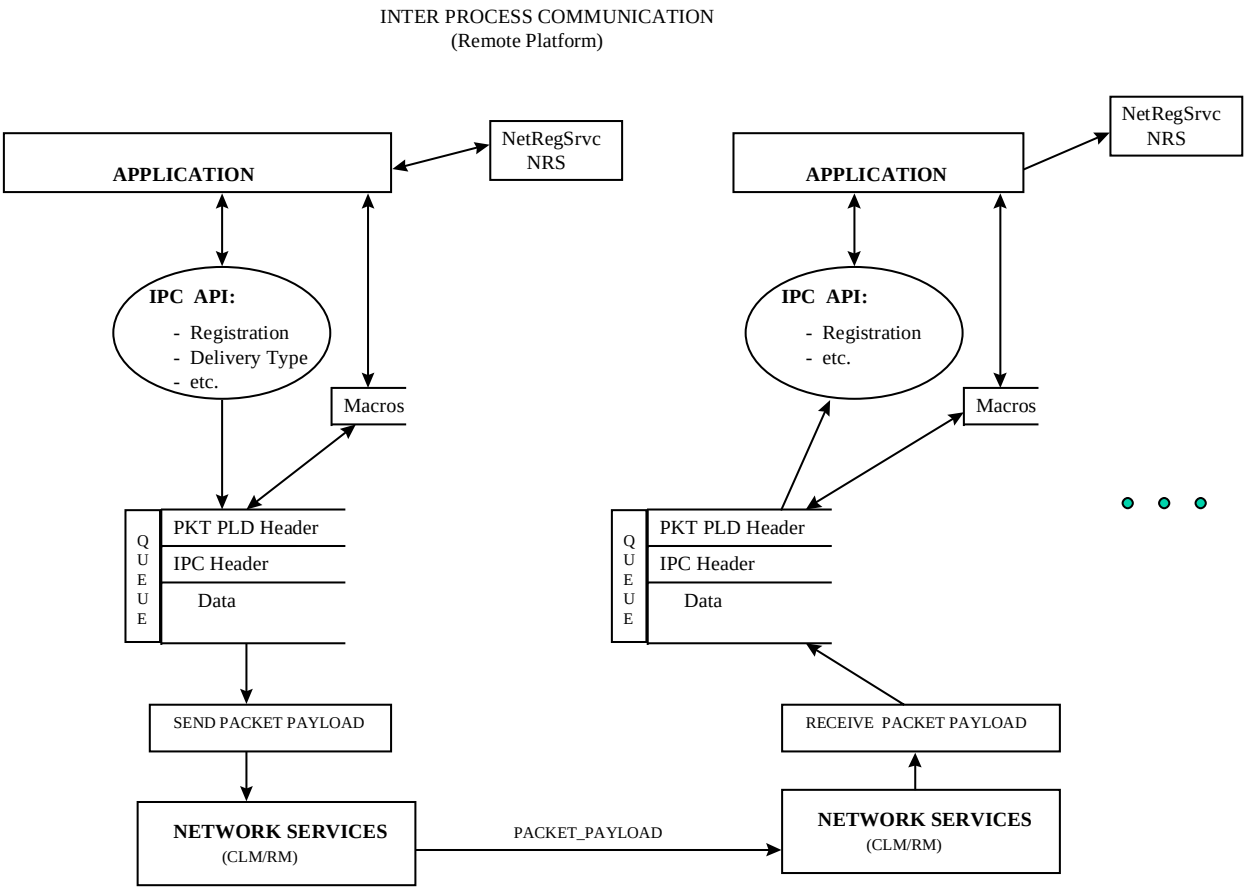


Figure 2. Inter-Node Communications

1.2 IPC Services Specifications

1.2.1 IPC Services Groundrules

IPC has the following assumptions and constraints:

- IPC will be started by Initialization & Termination Services.
- Routing of application data between networks (DCN to RTCN or vice-versa) will be performed by the applications and not by IPC.
- **IPC Services events are limited in size by the message queue. Message size is a UNIX tunable parameter that is currently set for approximately 32,000 bytes.**
- **Applications using IPC Services specify the destination of a message with a service ID (SID). A SID is a null terminated character string of up to 64 characters.**
- **Each time an application registers with IPC Services a message queue is created and each application is limited to 3 message queues. This maximum can be changed by modifying a parameter in an include file.**

1.2.2 IPC Services Functional Requirements

The Functional Requirements for IPC are arranged in the following major/minor functions:

1. Register/Deregister
2. Send Events
3. Receive Events

1 Registration / Deregistration

1.1 IPC will register an application with IPC upon request.

1.2 IPC will deregister an application with IPC upon request.

1.3 IPC will deregister a terminated application if the application did not deregister prior to termination.

2 Send Events

2.1 IPC will provide the capability to generate/build an event for point-to-point communication.

2.2 IPC will acquire the information for generation and distribution of the Packet Payload as per the RTPS Packet Payload ICD.

2.3 IPC will provide the capability to deliver events between application processes within the same platform (intranode communication).

2.4 IPC will provide the capability to multicast the same event onto the network if the recording flag is set.

2.5 IPC will provide the capability to deliver events between application processes on different platforms (internode communication).

2.6 IPC will provide the capability for an application to send an event to an application residing in a specific platform.

2.7 IPC will provide the capability to send an event without specifying the receiver's platform address **(location transparency).**

2.8 IPC will support the following delivery methods:

a) Point-to-point

b) Multipoint or multicast.

2.9 IPC will allow an application process to specify the following for a send event request:

a) Blocking – wait on completion of request processing.

b) Non-blocking – immediately return control to sender.

2.10 IPC will allow an application to specify the amount of time in milliseconds that a send request will wait before timing out on a blocking and *non-blocking* request.

2.11 IPC will provide error notification to the requesting application when an error is encountered in processing a send event request.

3 Receive Events

3.1 IPC will provide the capability to receive events from other application processes residing on the same platform **(intranode communication).**

3.2 IPC will provide the capability to receive events from other application processes residing on a different platform **(internode communication).**

3.3 IPC will allow an application process to specify the following for a receive event request:

a) Blocking – wait until an event is received.

b) Non-blocking – immediately return control to the receiver.

3.4 IPC will provide the capability for an application process to specify the amount of time in milliseconds that the application process will wait when a blocked or *non-blocked* receive event request is issued.

3.5 IPC will provide error notification to the receiver when an error is encountered in processing a receive event request.

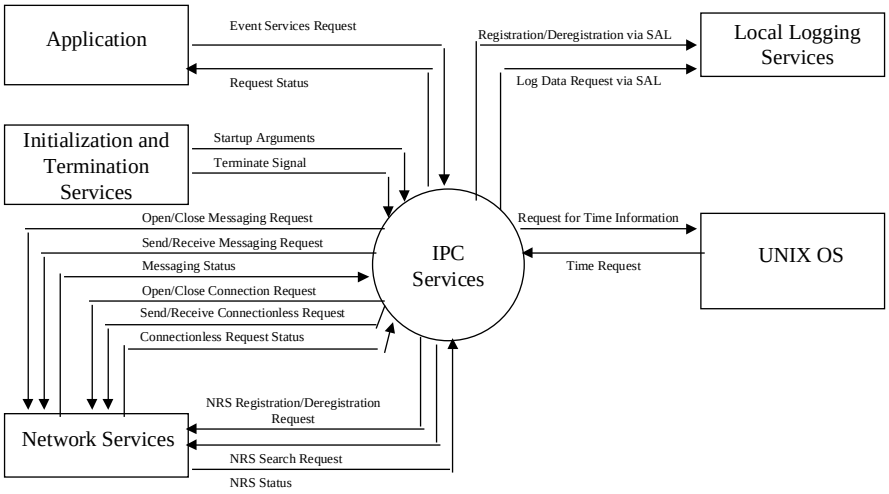
1.2.3 IPC Services Performance Requirements

This section is TBD.

1.2.4 IPC Services Interfaces Data Flow Diagrams

This section provides a description and diagram of all of the interfaces to IPC Services.

IPC Services Data Flow Diagram



IPC Services contains interfaces with Applications, Initialization and Termination Services, Network Services, Local Logging Services, Timing Services (post-Redstone), and System Messaging Services.

Applications register and de-register with IPC Services and use its API's to send and receive messages.

Initialization and Termination Services starts the es_man process and terminates it when requested.

Network Services handles the setup and breakdown of connection requests. It also provides the lower level network read and write capabilities.

Timing Services will provide time information (post-Redstone). IPC Services will use UNIX system calls to receive time information for Redstone.

1.3 IPC Services Design Specification

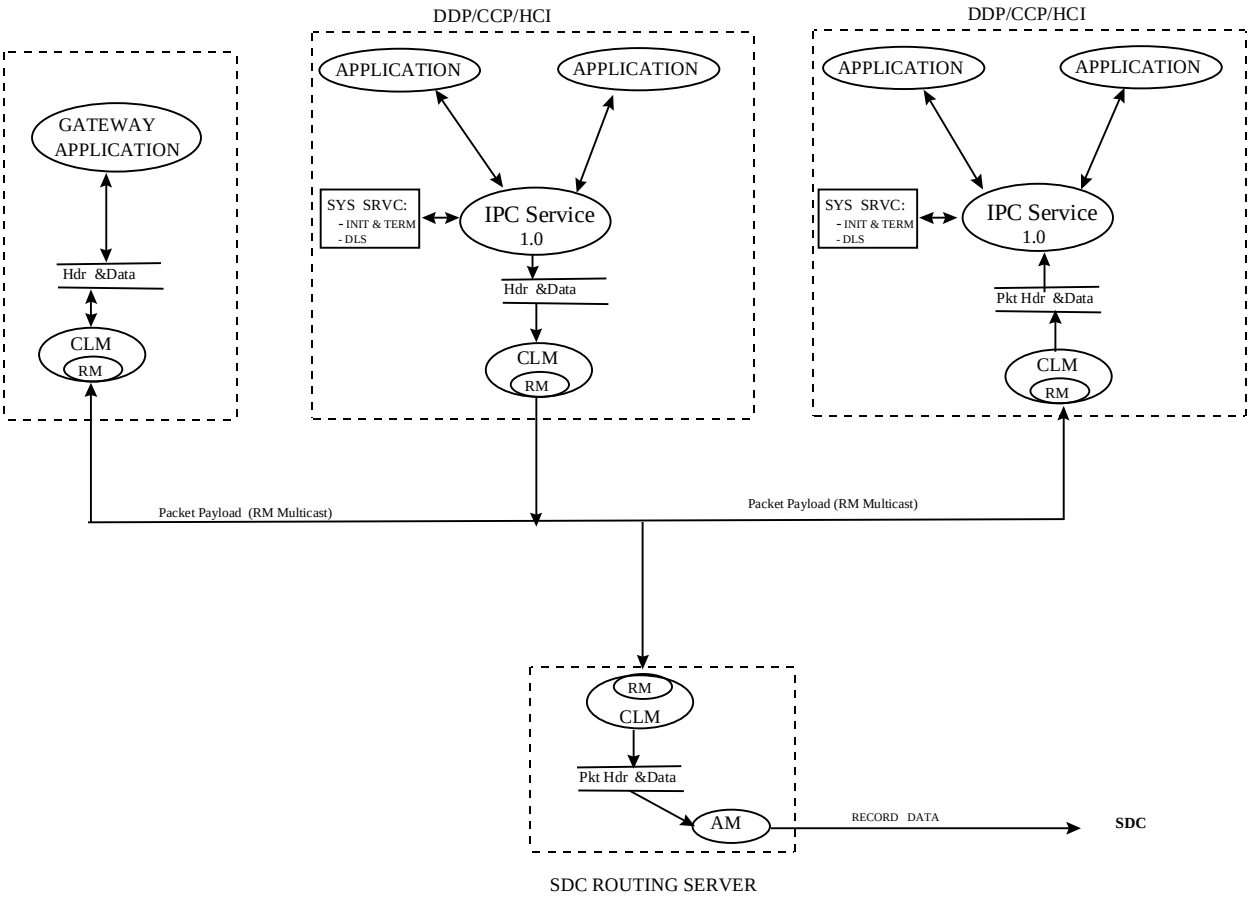
IPC is comprised of two daemon processes, es_man and es_lan_man, a a number of API's used by applications to use the services. The es_man process handles registrations, deregistrations, and cleanup when a process terminates without deregistering. The es_lan_man process main function is to provide application data distribution by calling the appropriate Network APIs for remote communications. It also handles local process-to-process communication.

1.3.1 IPC Services Detailed Data Flow

1.3.1.1 IPC Services Detailed Data Flow Diagram – Level 1

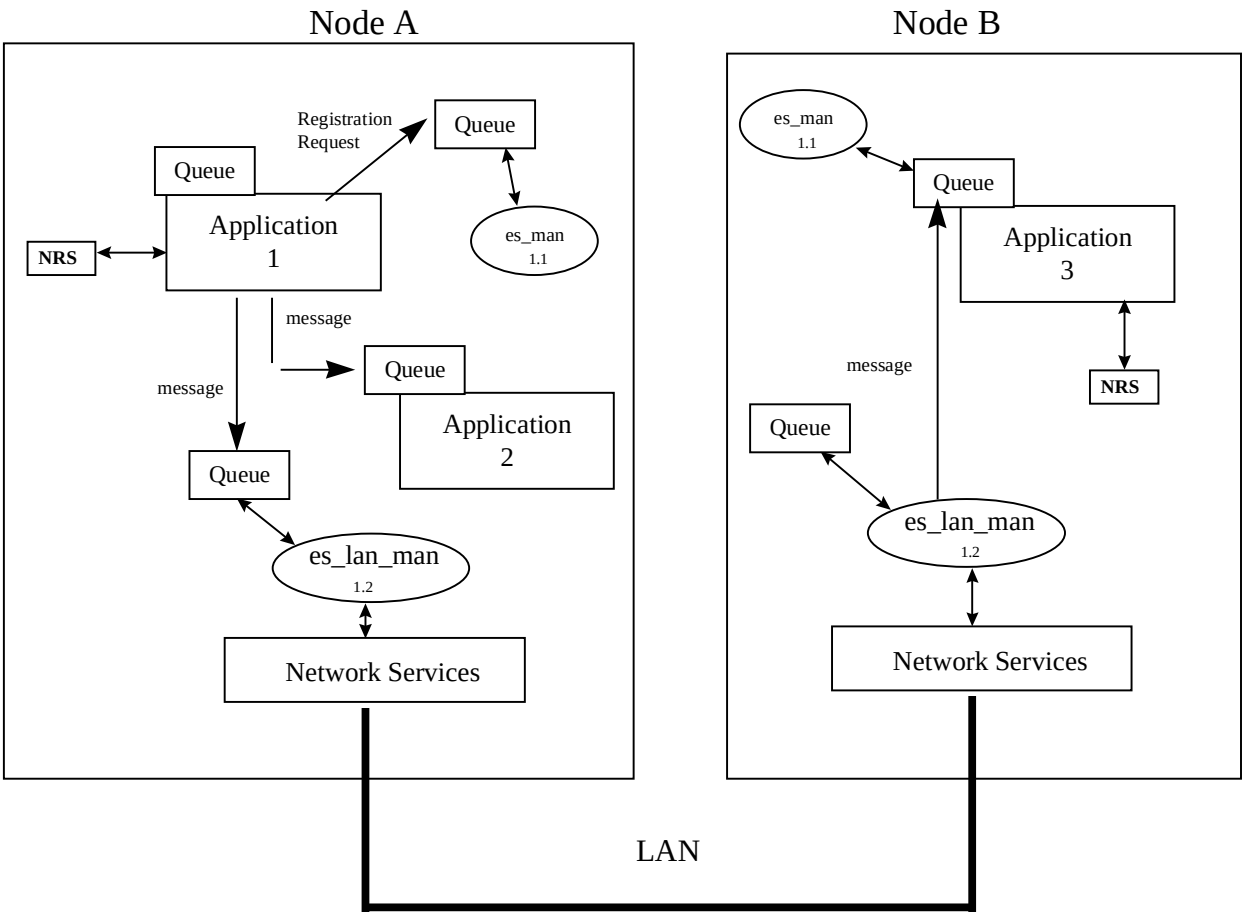
This data flow provides a pictorial representation of the data flow between external sources and destinations and the major and minor functions of IPC Services.

DATA FLOW DIAGRAM - LEVEL 1



1.3.1.2 IPC Services Detailed Data Flow Diagram - Level 2

IPC Services Functional Data Flow Diagram



1.3.2 IPC Services External Interfaces

1.3.2.1 Local Log Messages

IPC Services will use Local Logging Services to store messages.

1.3.2.2 IPC Services Display Formats

IPC Services has one graphical interface, es_diag, a diagnostic tool.

1.3.2.3 IPC Services Input Formats

This section is inapplicable to IPC.

1.3.2.4 IPC Services Recorded Data

This section is inapplicable to IPC.

1.3.2.5 IPC Services Printer Formats

This section is inapplicable to IPC.

1.3.2.6 Interprocess Communications (IPC)

The following diagrams show the structure of IPC Services events

1.3.2.6.1 IPC Services Messages

The following diagram shows the structure of the standard IPC Services message.

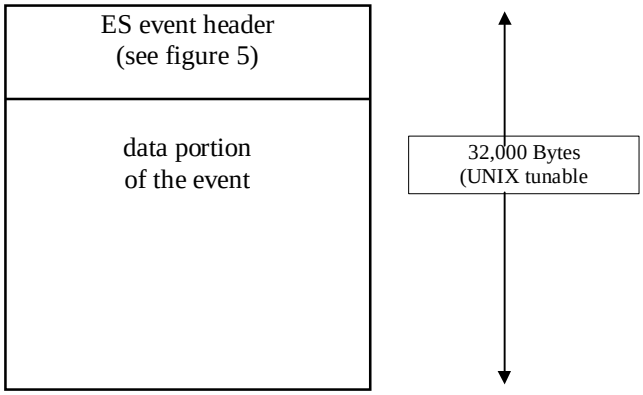


Figure 3. Intra-node message layout

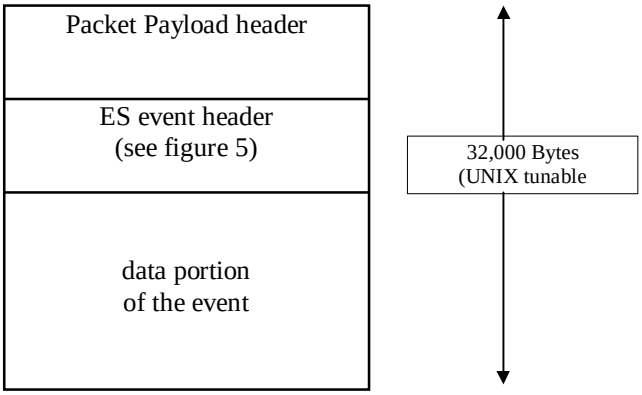


Figure 4. Inter-node message layout[1]

[1] For Redstone only, the ES event header will be removed prior to sending the message on the network.

1.3.2.6.2IPC Services Header

The following diagram shows the structure of the standard IPC Services header. It is used on all point-to-point messages and on packets to pass them between IPC daemons within a node.

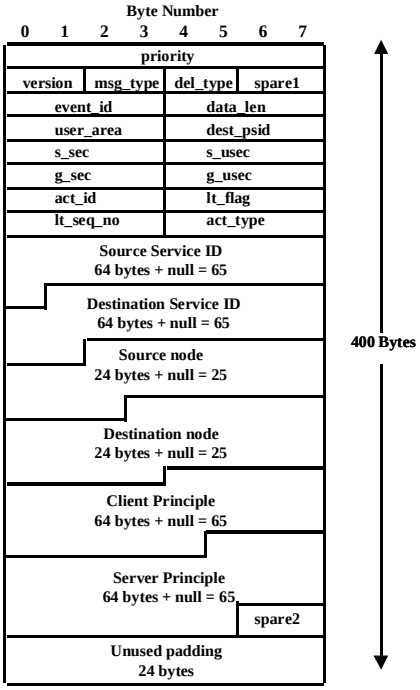
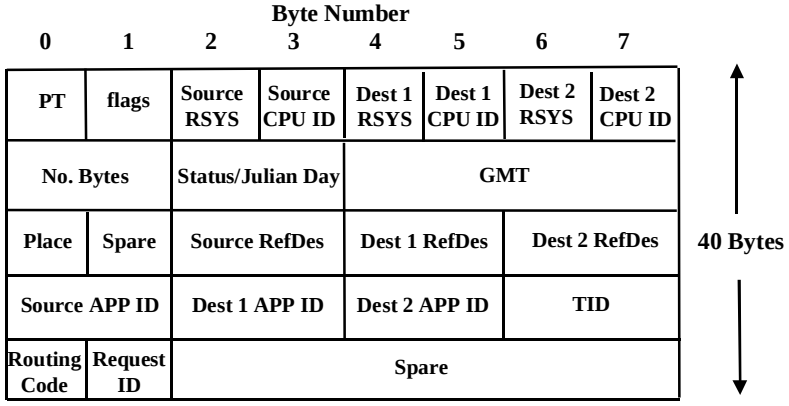


Figure 5. IPC Services Header

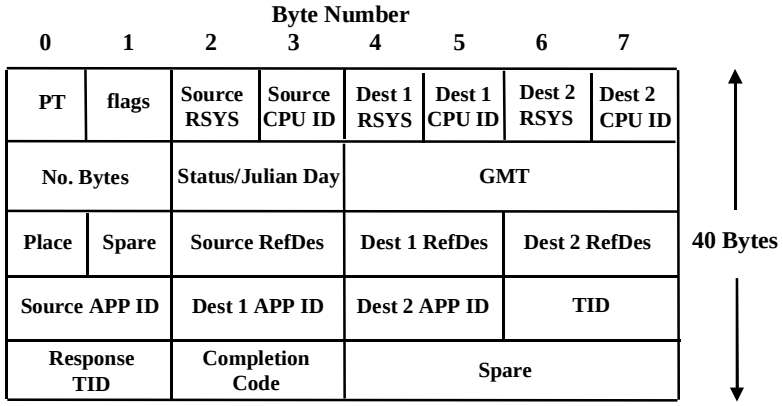
1.3.2.6.3 C-C Type 1 Packet Header

The following diagram shows the structure of the C-C Type 1 packet header. IPC is responsible for filling in some fields and provides macros to the application caller to fill in others.



1.3.2.6.4 C-C Type 0 Response Packet Header

The following diagram shows the structure of the C-C Type 0 Response packet header. IPC is responsible for filling in some fields and provides macros to the application caller to fill in others.



1.3.2.7 IPC Services External Interface Calls

This is the data that is sent to IPC Services modules via a calling mechanism (e.g., API call)

1.3.2.7.1 Register/Deregister Functions

1.3.2.7.1.1 The es_registration Function

Description:

The **es_register()** function allocates memory for the IPC handle, invokes the **es_q_create** method, registers the application with **NRS** --if the application requests, and registers the application with IPC.

Syntax: (ES_Handle *) es_register (char *service_id, int max_events, int flags)

1.3.2.7.1.2 The es_deregistration Function

Description:

The **es_deregister()** function deregisters the application from IPC, invokes the **es_q_destroy** method, deregisters the application from **NRS** --if the application was registered with **NRS**, and frees the memory allocated for the ES Handle.

Syntax: int es_deregister (ES_handle *hand)

1.3.2.7.1.3 The es_set_pl_opt Function

Description:

The **es_set_pl_opt()** function is used by the calling application to enable or disable the sending and receiving of packet using IPC. To utilize this service the application must be registered with IPC using *es_register* ().

Syntax: int es_set_pl_opt (ES_handle *hand, char *sid, ES_Psid *psid, ES_PL_Flag flag)

1.3.2.7.2 Send Functions

1.3.2.7.2.1 The es_send Function

Description:

The **es_send()** function is used by the calling application to send an event to another application To utilize this service, the application must be registered with IPC using *es_register* (). An event buffer must have been created and event control parameters such as destination SID must be set. The event data needs to be copied (or built) into the event buffer.

Syntax:

int es_send (ES_handle *hand, ES_Hdr *event, struct timeval *timeout, int *fail_num, ES_Error error_struct[])

1.3.2.7.2.2 Allocate a Message Buffer

Description:

The ES_GET_EVENT _BUF macro allocates and initializes a IPC message buffer.

Syntax: (int) ES_GET_EVENT_BUF (int size, ES_Hdr *message, void *data)

1.3.2.7.2.3 Allocate a Packet Buffer

Description:

The ES_GET_PL _BUF macro allocates and initializes a IPC message buffer.

Syntax: (int) ES_GET_PL_BUF (int size, ES_Hdr *message, void *data)

1.3.2.7.2.4 Initialize an Message Buffer

Description:

The ES_SET_DEFAULTS macro initializes a IPC message buffer.

Syntax: (int) ES_SET_DEFAULTS (ES_Hdr *header)

1.3.2.7.2.5 Initialize an Packet Buffer

Description:

The ES_SET_PL_DEFAULTS macro initializes a IPC packet buffer.

Syntax: (int) ES_SET_PL_DEFAULTS (ES_Hdr *packet)[2]

1.3.2.7.2.6 Write public data to a message buffer

Description:

These ES_SET macros allow applications to write data to an ES_Hdr structure.

ES_SET_DEST_NODE	Sets the destination node field,
ES_SET_DEST_SID	Sets the destination service ID field,
ES_SET_DEST_PSID	Sets the multicast connection number for multicast sends,
ES_SET_PRIORITY	Sets the priority field,
ES_SET_DATA_LEN	Sets the data length field,
ES_SET_DEL_TYPE	Sets the delivery type for the message,
ES_SET_EVENT _ID	Sets the event ID field,
ES_SET_U_AREA	Sets the user area field,
ES_SET_CPRINC	Sets the client principle field for Kerberos messages, ES_SET_SPRINC

Syntax: ES_SET… (ES_Hdr *header, val)

1.3.2.7.2.7 Write Public Data to a IPC Packet

Description:

These ES_SET_PL macros allow applications to write data to an ES_Hdr structure.

ES_SET_PL_TYPE	Sets the payload type field,
ES_SET_PL_LOGGING	Sets the payload logging bits,
ES_SET_PL_RESP_XPECT	Sets the payload response expected flag,
ES_SET_PL_DEST1_RESP_SYS	Sets the payload responsible system for destination 1,
ES_SET_PL_DEST1_CPU_ID	Sets the CPU_ID for destination 1,
ES_SET_PL_DEST1_REFDES	Sets the payload refdes field for destination 2,
ES_SET_PL_DEST_APP_ID	Sets the application ID for destination 1,
ES_SET_PL_DEST2_RESP_SYS	Sets the payload responsible system for destination 2,

[2] The LOGGING Bits will default to recording (i.e., request logging to SDC) unless specifically set/reset by the ES_SET_PL_LOGGING macro call.

ES_SET_PL_DEST2_CPU_ID	Sets the CPU_ID for destination 2,		
ES_SET_PL_DEST2_REFDES	Sets the payload refdes for destination 1,	ES_SET_PL_DEST2_APP_ID	Sets the pa
ES_SET_PL_RESPONSE	Sets the payload response filed,		
ES_SET_PL_ROUTE_CODE	Sets the payload routing code,		
ES_SET_PL_REQUEST_ID	Sets the payload request ID,		
ES_SET_PL_RESP_TRAN_ID	Sets the payload response transaction ID,		
ES_SET_PL_COMP_CODE	Sets the payload completion code		

Syntax: ES_SET_PL… (ES_Hdr *packet, val)

1.3.2.7.2.8 Copy data into a packet structure

Description:

The ES_SET_PL_DATA macro allows applications to initialize an ES_Hdr structure for packets.

Syntax: ES_SET_PL_DEFAULT (ES_Hdr*packet, void *packet_data, in data_size)

1.3.2.7.3 Receive Functions

1.3.2.7.3.1 The es_receive Function

Description:

The **es_receive()** function is used by the calling application to receive an event from another application To utilize this service, the application must be registered with IPC using *es_register* ().

Syntax:

int es_receive (ES_handle *hand, ES_Hdr *event, struct timeval *timeout)

1.3.2.7.3.2 Retrieve Public Data from the IPC Messages

Description:

These ES_GET macros allow applications to retrieve data from an ES_Hdr structure.

ES_SET_DEST_NODE	Sets the destination node field,
ES_SET_DEST_SID	Sets the destination service ID field,
ES_SET_DEST_PSID	Sets the multicast connection number for multicast sends,
ES_SET_PRIORITY	Sets the priority field,
ES_SET_DATA_LEN	Sets the data length field,
ES_SET_DEL_TYPE	Sets the delivery type for the message,
ES_SET_EVENT _ID	Sets the event ID field,
ES_SET_U_AREA	Sets the user area field,
ES_SET_CPRINC	Sets the client principle field for Kerberos messages, ES_SET_SPRINC
ES_GET_DATA_PTR	Gets a pointer to the data portion of the message,

Syntax: ES_GET_PL… (ES_Hdr *message)

1.3.2.7.3.3 Retrieve Public Data from the IPC Packets

Description:

These ES_GET_PL macros allow applications to retrieve data from an ES_Hdr structure.

ES_GET_PL_DATA_PTR Gets a pointer to the data portion of the packet,
ES_GET_PL_RESPONSEGets the payload response field, ES_GET_PL_DEST1_RESP_SYS, Gets the responsible system for destination 1, ES_GET_PL_DEST1_CPU_ID Gets the CPU_ID for destination 1,
ES_GET_PL_REFDES1 Gets the refdes for destination 1,
ES_GET_PL_DEST2_RESP_SYS, Gets the responsible system for destination 2,
ES_GET_PL_DEST2_CPU_ID Gets the CPU_ID for destination 2,
ES_GET_PL_REFDES2 Gets the refdes for destination 2,
ES_GET_PL_APP_ID Gets the payload application ID,
ES_GET_PL_ROUTE_CODE Gets the payload routing code,
ES_GET_PL_REQUEST_ID Gets the payload request ID,
ES_GET_PL_TRAN_ID Gets the payload transaction ID,
ES_SET_PL_COMP_CODE Gets the payload completion code, ES_GET_PL_SOURCE_RESP_SYS Gets the pa application ID

Syntax: ES_GET_PL… (ES_Hdr *packet)

1.3.2.8 IPC Services Table Formats

This section is inapplicable to IPC Services.

1.3.3 IPC Services Test Plan

IPC Services system-level tests may be run in either or both the IDE or SDE environments. These tests are run on the basic HCI, CCP or DDP platforms. There are no special hardware configurations required.

IPC Services testing also requires a CLCS application or a CLCS like-application test tool that exercises the point-to-point and multicast delivery methods.

The specific test cases that will be run include:

1. [sending IPC] Multicast send from the HCI to CCP and SDC
2. [sending IPC] Multicast send from the CCP to a Gateway and SDC
3. [sending IPC] Point-to-Point send within an SGI
4. [receiving an IPC] (Multicast) receive at the HCI from the DDP
5. [receiving an IPC] (Multicast) receive at the CCP from the Gateway
6. [command & response packet] Multicast send from the HCI -> CCP -> Gateway;
   Gateway response to CCP -> HCI.